EE631 Cooperating Autonomous Mobile Robots

Lecture 3: Path Planning Algorithm

Prof. Yi Guo

ECE Dept.

Plan

- Representing the Space
- Path Planning Methods
- A* Search Algorithm
- D* Search Algorithm

Representing Space

- The most natural representation of a robot's environment is a map
- A robot's internal representation of its space is typically required for at least three different classes of tasks:
  - To establish what parts of the environment are free for navigation
  - To recognize regions or locations in the environment
  - To recognize specific objects within the environment

Spatial Decomposition

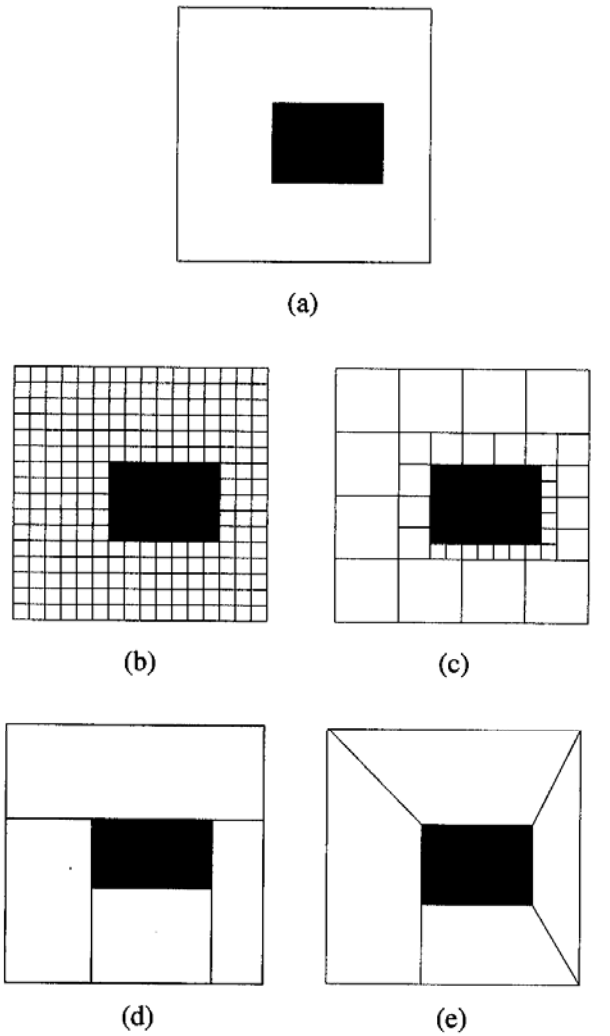


Figure 5.1. An environment and five different spatial decompositions of it. (a) Sample environment. (b) Uniform. (c) Quadtree. (d) BSP. (e) Exact.

Geometric Representation

- Geometric maps are composed of the union of simple geometric primitives
  - 2D map:
    - Points
    - Lines, line segments and polylines (piecewise linear curves)
    - Circles and arcs of circles
    - Polynomials
    - Splines

Geometric Representation

- 3D map:
  - Points
  - Planar surfaces
  - Surface patch networks
  - Circles and ellipsoids
  - Spline surfaces

Topological Representation

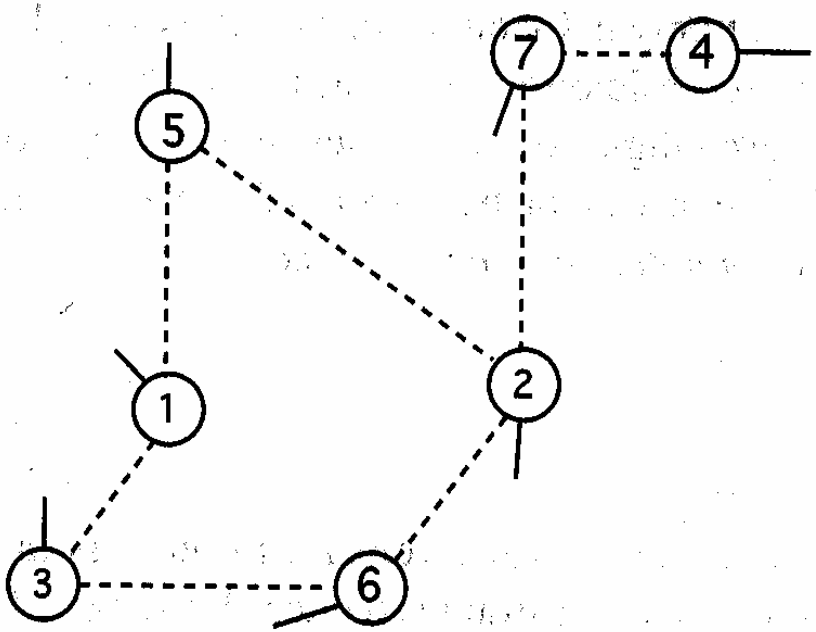
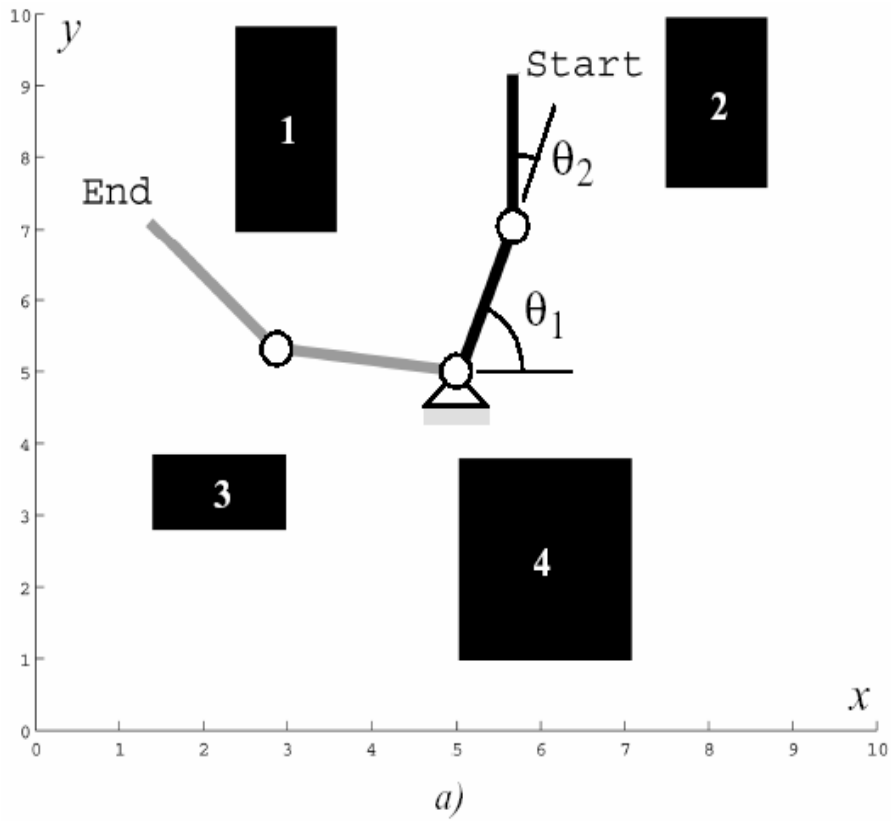
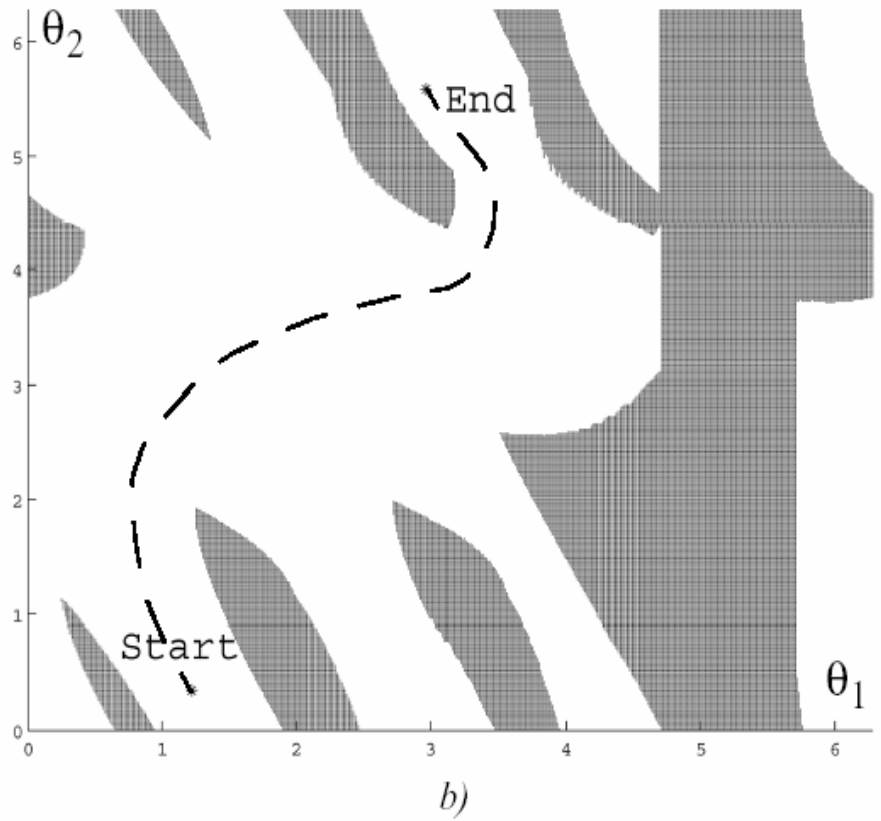


Figure 5.4. Graph-based representation of a robotic environment. Vertices correspond to known landmarks and edges to the paths between them.

Path Planning: Configuration Space

- State or configuration q can be described with k values q_i



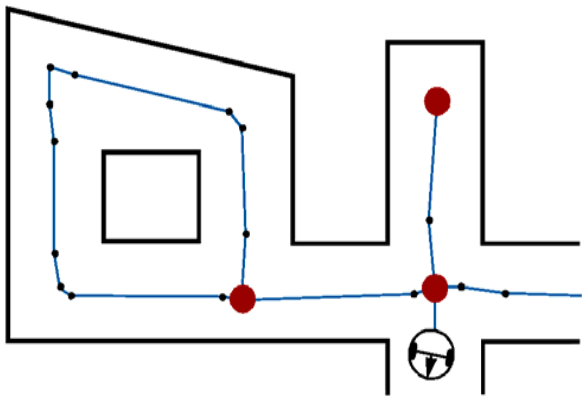


- What is the configuration space of a mobile robot?

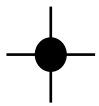
Path Planning Overview

1. Road Map, Graph construction

- Identify a set of routes within the free space



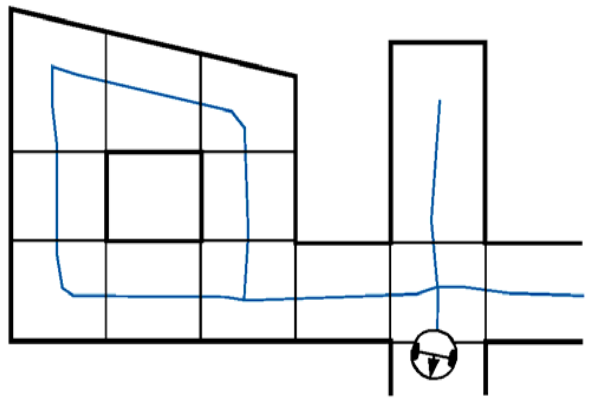
- Where to put the nodes?
- Topology-based:
  - at distinctive locations



- Metric-based:
  - where features disappear or get visible

2. Cell decomposition

- Discriminate between free and occupied cells

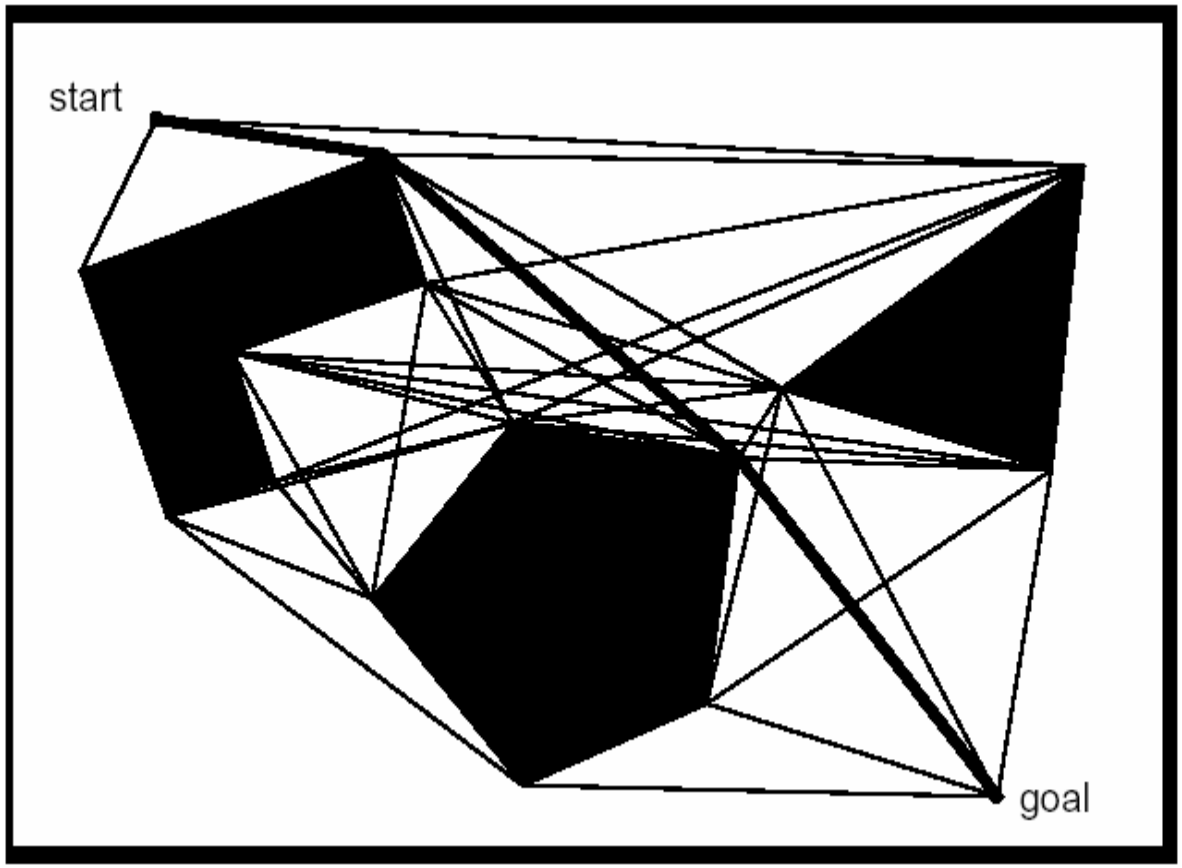


- Where to put the cell boundaries?
- Topology- and metric-based:
  - where features disappear or get visible

3. Potential Field

- Imposing a mathematical function over the space

Road-Map Path Planning: Visibility Graph

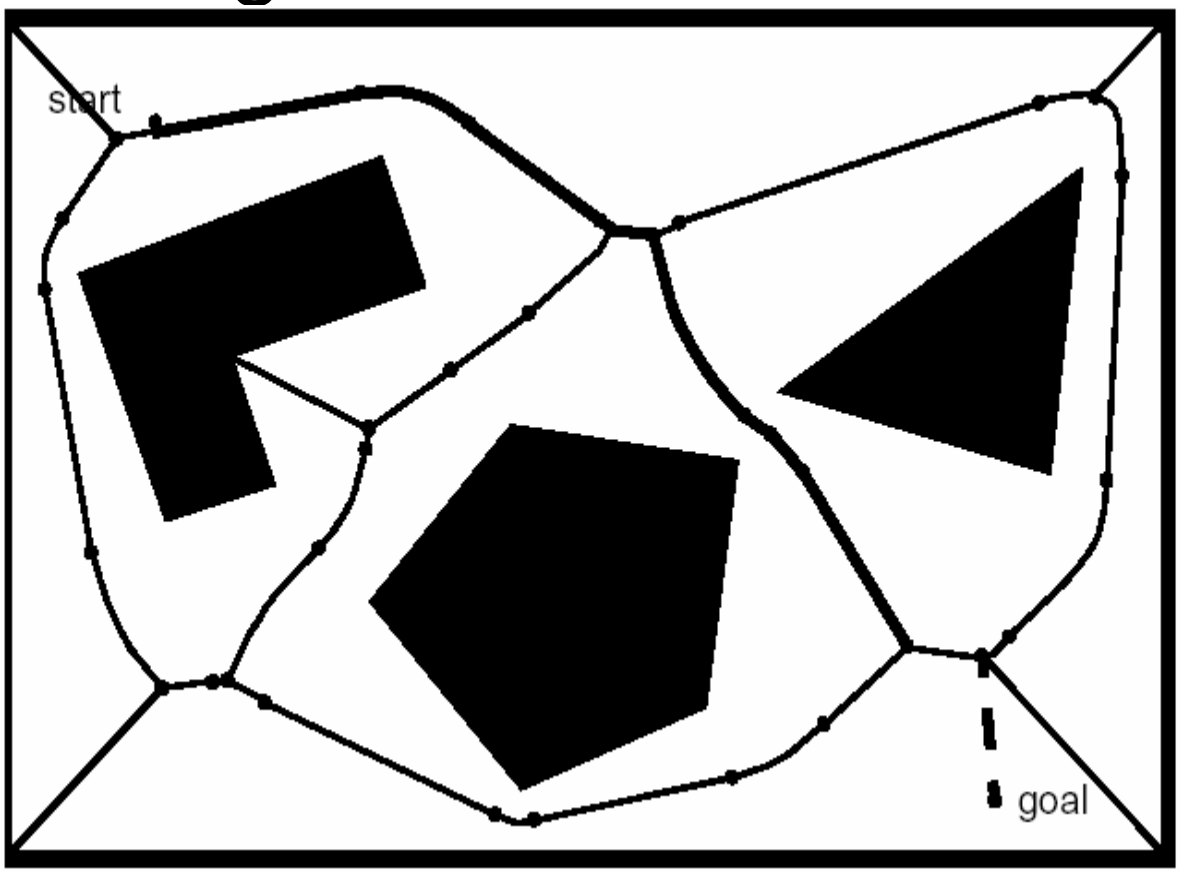


The visibility graph for a polygonal configuration space consists of edges joining all pairs of vertices that can see each other (including the initial and goal positions).

Path planner is to find the shortest path from the initial position to the goal position along the roads defined by the visibility graph.

- Shortest path length
- Grow obstacles to avoid collisions

Road-Map Path Planning: Voronoi Diagram



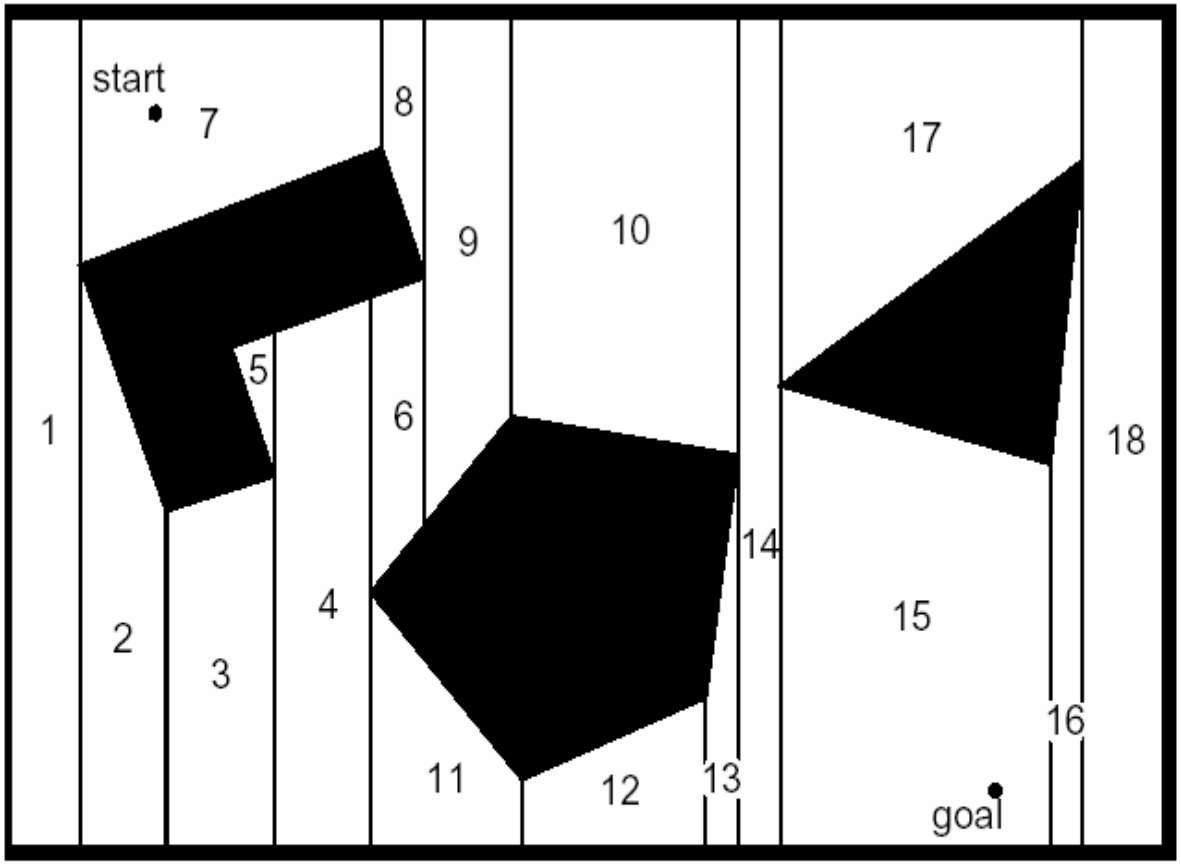
For each point in the free space, compute its distance to the nearest obstacles; At points that are equidistant from two or more obstacles, such a distance plot has sharp ridges; The Voronoi diagraph consists of the edges formed by theses sharp ridge points.

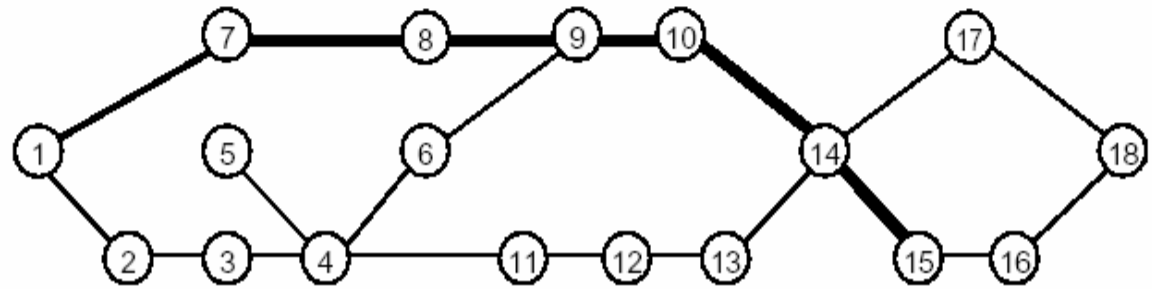
- Easy executable: Maximize the sensor readings
- Works also for map-building: Move on the Voronoi edges

Road-Map Path Planning: Cell Decomposition

- Divide space into simple, connected regions called cells
- Determine which open sells are adjacent and construct a connectivity graph
- Find cells in which the initial and goal configuration (state) lie and search for a path in the connectivity graph to join them.
- From the sequence of cells found with an appropriate search algorithm, compute a path within each cell.
  - e.g. passing through the midpoints of cell boundaries or by sequence of wall following movements.

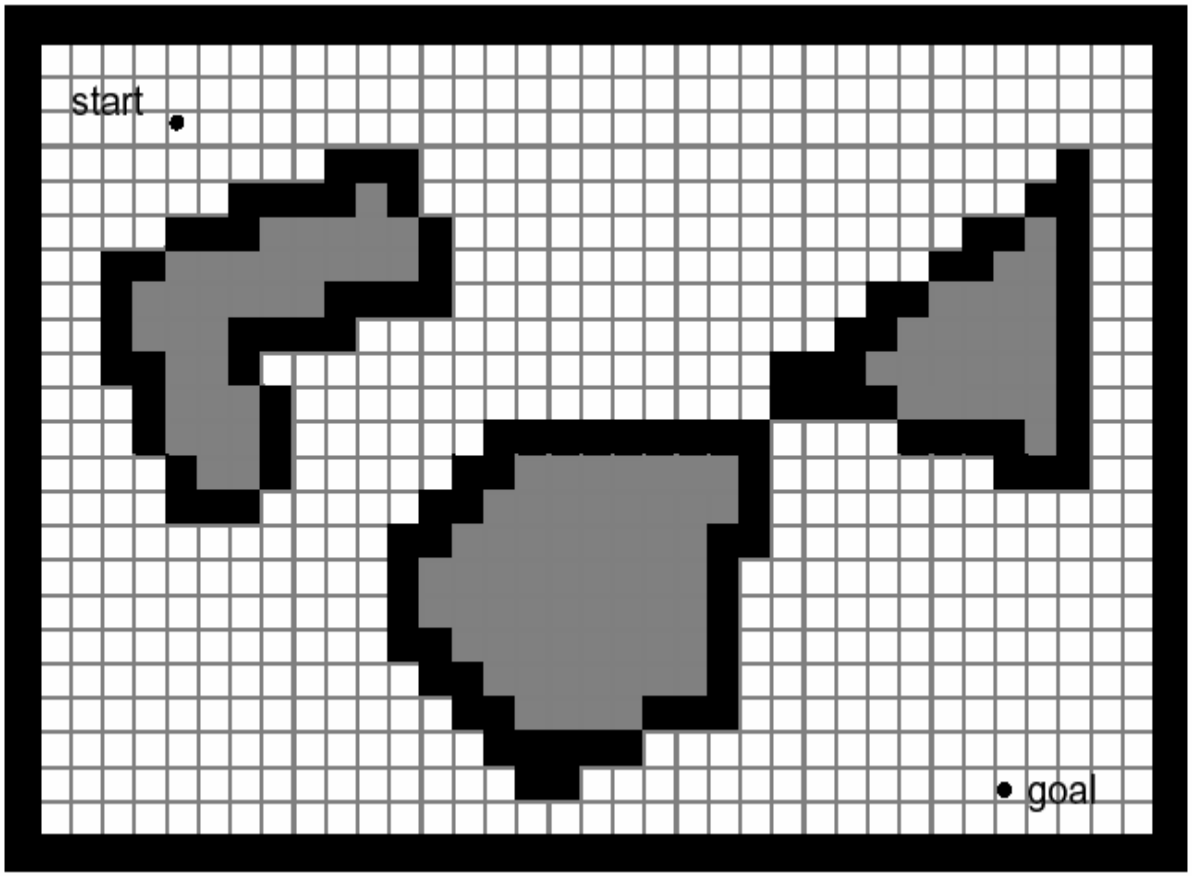
Road-Map Path Planning: Exact Cell Decomposition





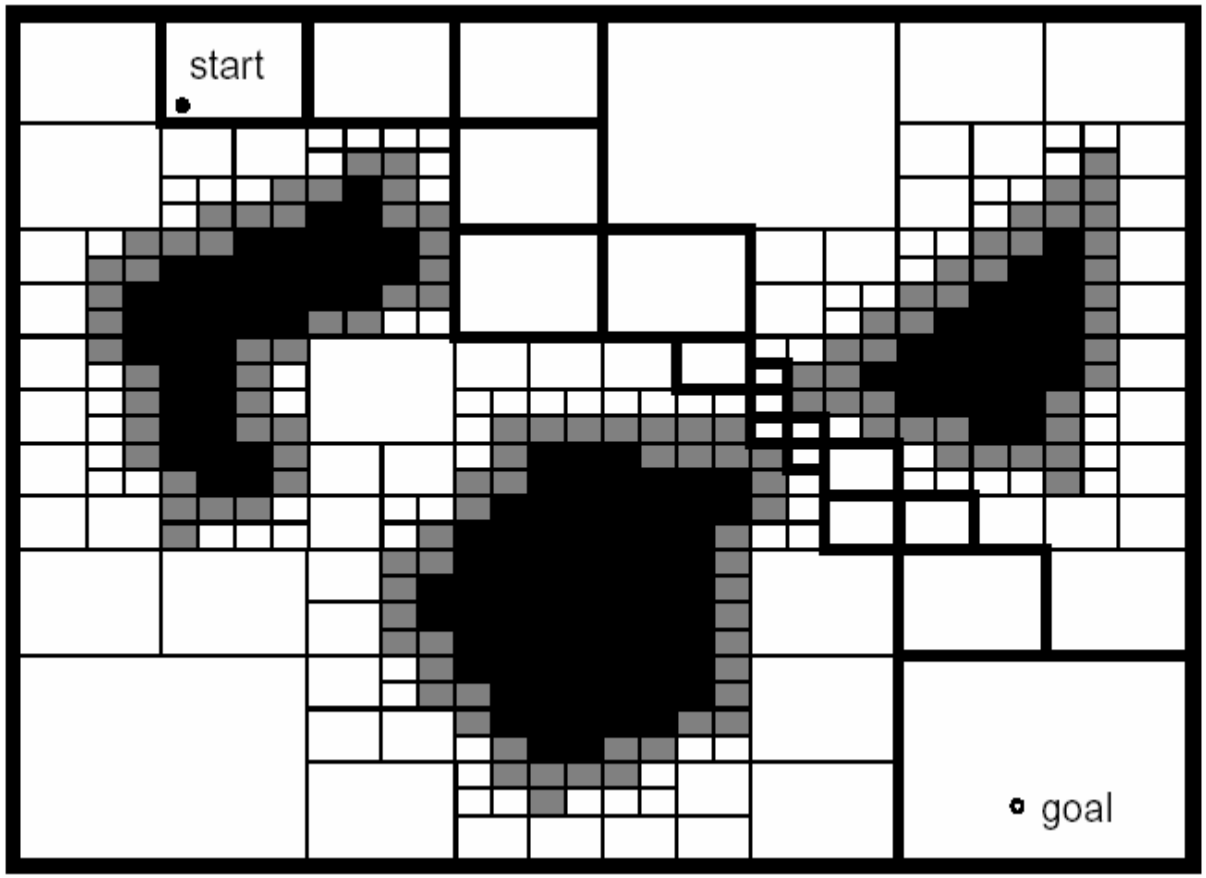
The cells are each either completely free or completely occupied.

Road-Map Path Planning: Approximate Cell Decomposition

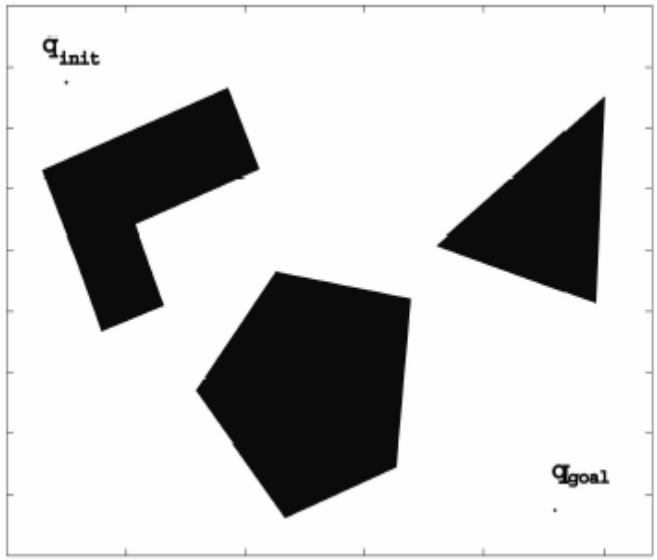


Fixed-size cell decomposition such as grid-based representation.

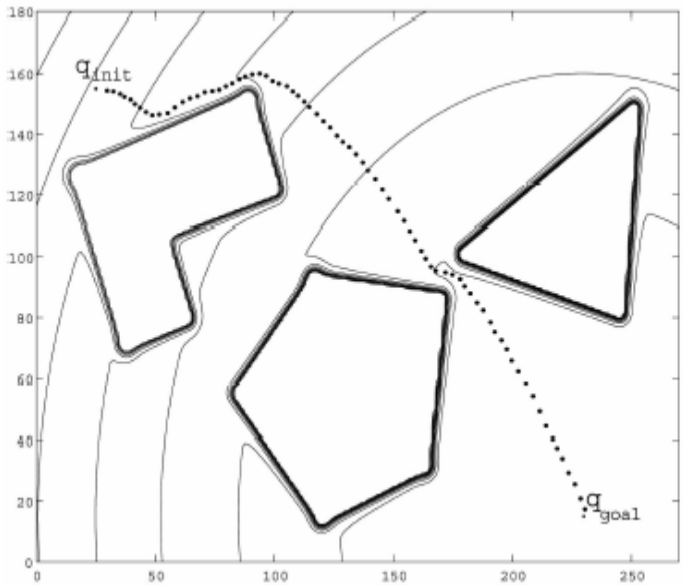
Road-Map Path Planning: Adaptive Cell Decomposition

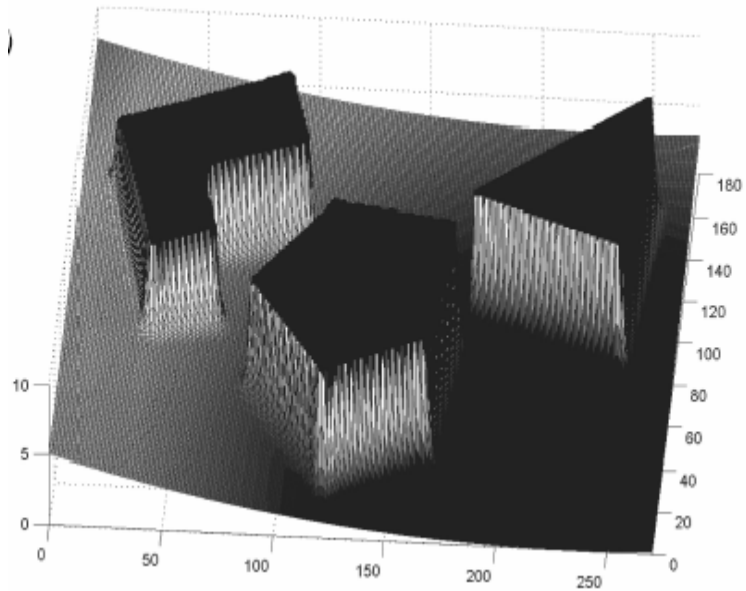


Potential Field Path Planning



- Robot is treated as a *point under the influence* of an artificial potential field.
  - Generated robot movement is similar to a ball rolling down the hill
  - Goal generates attractive force
  - Obstacle are repulsive forces





Potential Field Path Planning: Potential Field Generation

- Generation of potential field function *U(q)*
  - attracting (goal) and repulsing (obstacle) fields
  - summing up the fields
  - functions must be differentiable
- Generate artificial force field *F(q)*

$$F(q) = -\nabla U(q) = -\nabla U_{att}(q) - \nabla U_{rep}(q) = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix}$$

- Set robot speed (v_x, v_y) proportional to the force *F*(*q*) generated by the field
  - the force field drives the robot to the goal
  - if robot is assumed to be a point mass

Potential Field Path Planning: Attractive Potential Field

- Parabolic function representing the Euclidean distance $\|q - q_{goal}\|$ to the goal

$$U_{att}(q) = \frac{1}{2} k_{att} \cdot \rho^2_{goal}(q)$$

- Attracting force converges linearly towards 0 (goal)

$$\begin{aligned} F_{att}(q) &= -\nabla U_{att}(q) \\ &= -k_{att} \cdot \rho_{goal}(q) \nabla \rho_{goal}(q) \\ &= -k_{att} \cdot (q - q_{goal}) \end{aligned}$$

Repulsing Potential Field

- Should generate a barrier around all the obstacle
  - strong if close to the obstacle
  - not influence if far from the obstacle

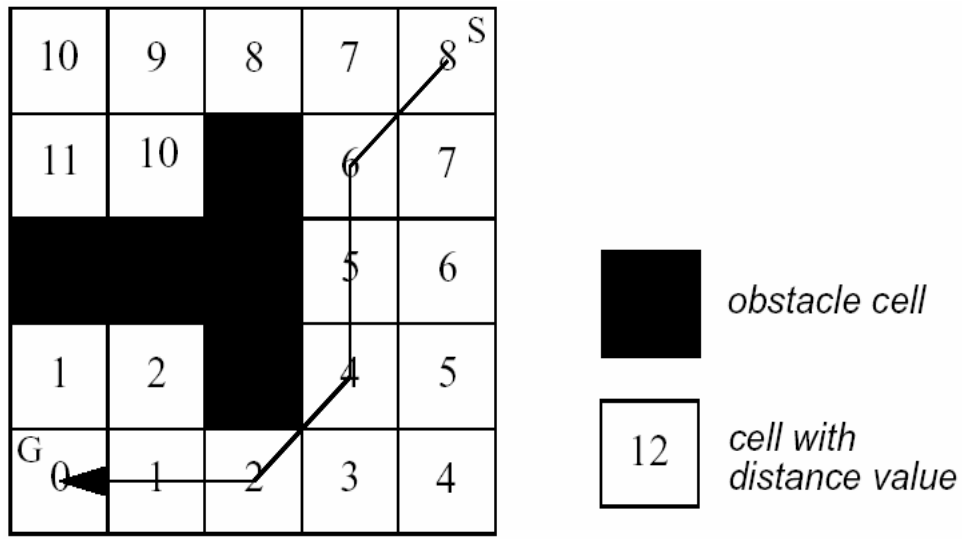
$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0} \right)^2 & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) \ge \rho_0 \end{cases}$$

  - $\rho(q)$: minimum distance to the object
  - Field is positive or zero and *tends to infinity* as q gets closer to the object

$$F_{rep}(q) = -\nabla U_{rep}(q) = \begin{cases} k_{rep} \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2(q)} \frac{q - q_{goal}}{\rho(q)} & \text{if } \rho(q) \le \rho_0 \\ 0 & \text{if } \rho(q) \ge \rho_0 \end{cases}$$

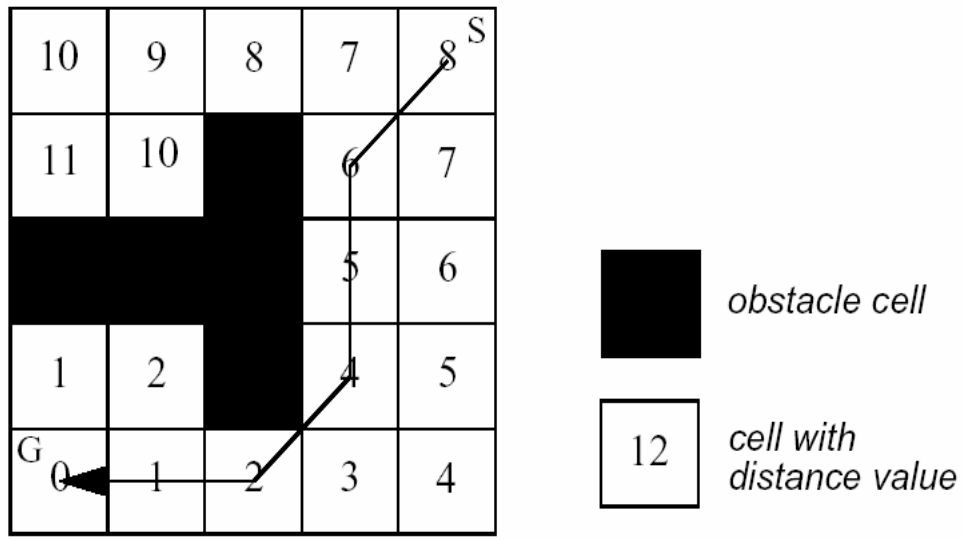
Road-Map Path Planning: Path / Graph Search Strategies

- Wavefront Expansion:
  - Considering a conductive material with heat radiating out from the initial node to the goal node
- Breadth-First Search
  - Expand the graph to all direct successors



Road-Map Path Planning: Path / Graph Search Strategies

- Depth-First Search
  - Generate the successors of a node just one at a time, and a trace is left at each node to indicate that additional operators can eventually be applied there if needed.
- Greedy search and A*
  - A classic method for computing optimal paths for holonomic robots

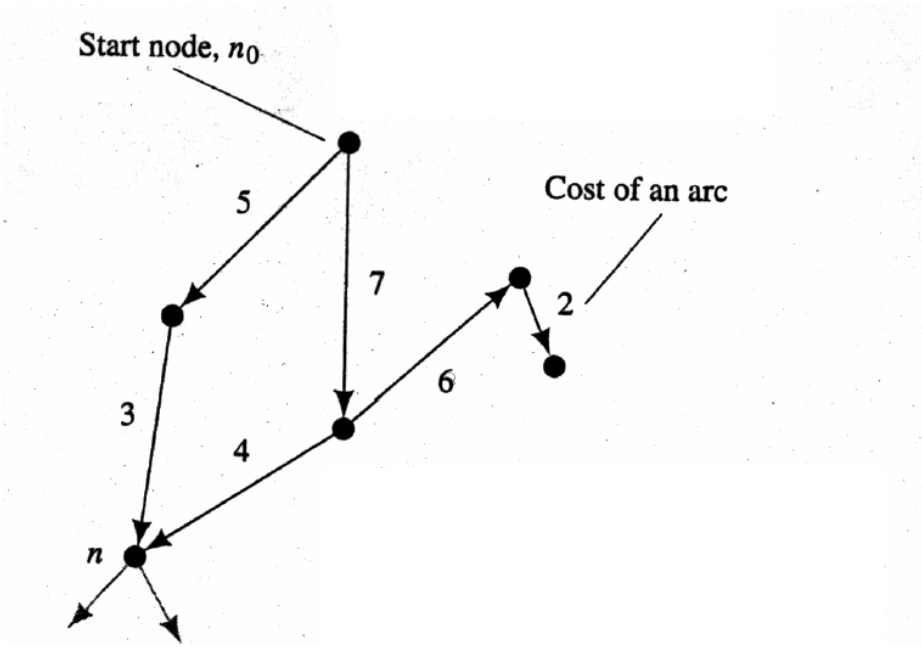


Path Planning Method: A* search

- Graph search -- A* search:
  - Evaluate node *n* by cost function:

    $f(n)=g(n)+h(n)$

    =estimated cost of the cheapest solution through *n*
  - Optimality guaranteed if h*(n)* never overestimates the cost to reach the goal



A* Search Algorithm

- Terms
  - State space: the space of all possible configurations for the task
  - Root node: initial starting point
  - Child nodes: places you can get to from a given configuration
  - OPEN list: places to explore that are connected to places you've been
  - CLOSED list: places you have already explored
  - Order the OPEN list by increasing cost
    - g( ) function: cost from root to the current node
    - h( ) function: estimated cost from the current node to the goal

A* Search Algorithm

- Put the root node on the OPEN list
- While the OPEN list is not empty, extract the first item
  - Test if the node is the goal
    - If it is, terminate the search
    - Return the path to the goal by linking through all the ancestors of the goal node
  - If it is not the goal, expand the node to find its children
  - For each child
    - If the child is on the CLOSED list
      - Carefully consider what needs to happen
      - If the node on the CLOSED list has a shorter g value, then delete the child
      - Else, make the current node the parent of the node on the CLOSED list and replace the g value of it with the g value of this child and propagate the g value to all of the descendents of this node

A* Search Algorithm

- If the child is on the OPEN list
  - Carefully consider what to do if it is
  - If the node on the OPEN list has a shorter g value, then delete the child
  - Else, delete the node on the OPEN list and insert the child in the proper order
- Else, if the child is not on either the OPEN or CLOSED lists
  - Put the child on the OPEN list with a link to its parent
- Place the current node on the CLOSED list

- If the OPEN list is empty, return failure in the search

Comparison of Search Methods

- Breadth-first search
  - Estimate of cost to the goal is zero
- Greedy/depth-first search
  - Estimate of cost from the root node is zero
- A* is optimal if the h function is exactly the cost to the goal
- A* returns an optimal path if the h function under-estimates the distance to the goal
- A* is pretty good if the h function only sometimes over-estimates the distance to the goal

Navigation Using A*

1. Generate a regular grid of cells that represents the robot's workspace
2. Place any known obstacles in the map (with high priority)
3. Plan a path to the goal using A*
4. Execute the path
5. If the environment can change,
   - update the map with each sensor reading
   - re-plan the complete path from the robot's current location to the goal using A*

Navigation Using A*

Now assume we are getting sensor data 50 times per second (20ms per sonar)

- We may not have time to do a complete replanning after each sensor reading
- Do we necessarily have to update the whole map?
- Do we necessarily have to replan the entire path?

The D* (dynamic A*) allows us to replan only what we need to with each piece of sensor information.

The D* [Dynamic A*] Algorithm

Conceptually: D* works on a field of pointers in a semi-regular grid

- Each pointer is connected to the next grid point on the optimal path to the goal
- In a static environment, the robot just follows the pointers to get to the goal
- In a dynamic environment, we use the D* algorithm to keep the pointers current in the space between the robot and the goal.

D* Algorithm

- Differences/Additions from the A* algorithm
  - Maintain an explicit state for each grid point: state={OPEN,CLOSED,NEW}
  - Nodes are not created/deleted as the grid always exists
  - g(G, X) = path cost to the goal node from X, also written as g(X)
  - Each node X has a key value k(X) which is the minimum of all g(X) values a node has assumed since being placed on the OPEN list.
    - This is a lower bound on the distance to the goal from X
    - This is the major difference between D* and focused D*

D* Algorithm

- Nodes can be considered to be RAISE or LOWER states
  - RAISE state k(X) < g(G, X) -- the state has a higher cost to the goal than it once did
  - LOWER state k(X) = g(G, X) -- the state currently has its lowest cost to the goal
- kmin = minimum k(X) of all X on OPEN
- kold = previous kmin, if one exists

Readings:

- A. Stentz, “Optimal and efficient path planning for partially-known environments”, *Proceedings of 1994 IEEE International Conference on Robotics and Automation*, 8-13 May 1994, Page(s):3310 - 3317 vol.4.
- A. Stentz, “The Focussed D* Algorithm for Real-Time Replanning”, *Proceedings of the International Joint Conference on Artificial Intelligence*, August, 1995.